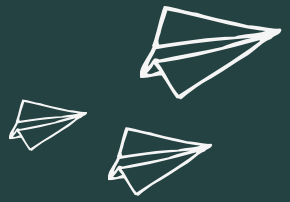


go 语 言 编 程

目录页

第五章

Part five

函数

第五章 函数

函数是一系列语句的集合，方便代码的重用和维护。

5.1 函数定义

函数的定义包括函数名、形参列表、返回值列表以及函数体。一般语法格式如下：

```
func 函数名称( [形参列表] ) [返回值列表]{
    函数体
}
```

5.2 可变参数

参数数量可变的函数称为可变参数的函数。在定义可变参数函数时，需要在参数列表的最后一个参数类型之前加上省略符号“...”。

5.3 多返回值

Go语言的函数可以有多个返回值。这个特性能够使得我们写出比其它语言更加优雅和简洁的代码。例如File.Read()函数可以同时返回读取的字节数和错误信息。

```
func (file *File) Read(b []Byte)(n int,err Error)
```

5.4 递归函数

程序调用自身的编程技巧称为递归。递归做为一种算法在编程语言中广泛应用。函数调用自身，称为递归函数。

5.5 错误处理

对于大多数函数而言，无法确保函数在任何时候都能成功运行，因为产生错误的原因很有可能超出程序员控制范围。

一般为如下模式：

```
func Foo(参数列表)(ret list, err error) {
// ...
}
```

调用时的处理模式：

```
n, err := Foo(参数)
if err != nil {
    // 错误处理
} else {
    //使用返回值 ret
}
```

5.6 defer

当一个函数调用前有关键字 defer 时, 那么这个函数的执行会推迟到包含这个defer 语句的函数即将返回前才执行。

defer 通常用于 open/close, connect/disconnect, lock/unlock 等这些成对的操作, 来保证在任何情况下资源都被正确释放。

5.6 defer

```go
func CopyFile(dst, src string) (w int64, err error) {
    srcFile, err := os.Open(src)
    if err != nil {
        return
    }
    defer srcFile.Close()
    dstFile, err := os.Create(dstName)
    if err != nil {
        return
    }
    defer dstFile.Close()
    return io.Copy(dstFile, srcFile)
}
```

这是一个文件拷贝的例子。即使其中的Copy()函数抛出异常， Go仍然会保证dstFile和srcFile会被正常关闭。一个函数中可以存在多个defer语句， defer语句的调用是遵照先进后出的原则，即最后一个defer语句将最先被执行。

5.6 defer

defer也可以跟一个用户自定义的匿名函数。

```
defer func() {
// 匿名函数的代码
} ()
```

5.6 defer

defer调用的函数参数的值在defer被定义时就确定了，而defer函数内部所使用的变量的值是在这个函数执行的时候才确定。

```go
func main() {
    i := 1
    defer fmt.Println("延时打印:", i) //i 是参数
    i++
    fmt.Println("常规打印:", i)
}
```

```go
func main() {
    i := 1
    defer func() {
        fmt.Println("延时打印", i) //i 是内部变量
    }()
    i++
    fmt.Println("常规打印:", i)
}
```

5.6 defer

defer 函数调用的执行时机是外层函数设置返回值之后，并且在即将返回之前。也就是说“return 返回值”语句并不是原子的。

```go
func double(x int) int {
    return x + x
}
func triple(x int) (r int) {
    defer func() {
        r += x
    }()
    return double(x)
}
func main() {
    fmt.Println(triple(3))
}
```

```go
// 等价与下面代码,打印结果是 9
func triple(x int) (r int) {
    // 1 设置返回值
    r = double(x)
    // 2 执行 defer 语句的函数
    func() {
        r += x
    }()
    // 3 函数返回
    return
}
```

5.7 panic、recover函数

Go语言追求简洁优雅，所以不支持传统的 try...catch...finally 这种异常处理结构。Go异常相关的关键字是：defer, panic, recover。 panic 是用来表示非常严重的不可恢复的错误的，一般会导致程序挂掉（除非recover）。Go语言对异常的处理可以这样简单描述：Go程序的执行过程中可以抛出一个panic的异常，然后在defer中通过recover捕获这个异常。

5.7 panic、recover函数

```go
func main() {
    defer func() { // 必须要先声明 defer，否则不能捕获到 panic 异常
        fmt.Println("c")
        if err := recover(); err != nil {
            fmt.Println(err) // 这里的 err 其实就是 panic 传入的内容
        }
        fmt.Println("d")
    }()
    foo()
}
func foo() {

    fmt.Println("a")
    panic("产生异常")
    fmt.Println("b")
    fmt.Println("f")
}
```

5.8 函数值

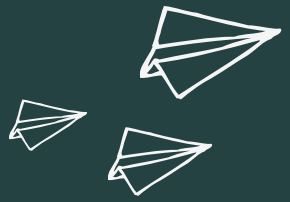
在Go中，函数被看作第一类值。函数值像其它值一样，拥有类型。函数类型可以使用type关键字定义，通过函数的参数和返回值区分。

函数值可以被赋值给其它变量，也可以作为其他函数的参数或返回值。对函数值的调用相当于对函数的调用。函数值属于引用类型，两个函数值之间不可比较。

5.9 闭包

闭包是可以包含自由变量的代码块，这些变量不在这个代码块内或者任何全局上下文中定义，而是在定义代码块的环境中定义。要执行的代码块为自由变量提供绑定的计算环境。

```
func Sqrt(num int) func() int { //变量num与匿名函数在同一环境中定义
        return func() int {
                num++
                return num * num
        }
}
func main() {
        s := Sqrt(0) //外部函数执行完成后，num并没有被销毁，值为0；
        fmt.Println(s()) //执行s函数，num的值为1，函数的返回值为1
        fmt.Println(s())//执行s函数，num的值为2，函数的返回值为4
        fmt.Println(s())//执行s函数，num的值为3，函数的返回值为9
}
```



THANKS

感谢聆听，期待反馈